Comprehensive Exercise Report

Team DIP392-marvels

Danush Vithiyarth Jaiganesh – 221ADB150
Meganath Nandhakumar – 221ADB061
Harikumaran Anandhan – 221ADB148
Ahmed Salah Kamal Abdelgawad Elsayed – 250ADB021
Alexander Fernando Thundiparambil Judy Thadevouse – 221AMB023

NOTE: You will replace all placeholders that are given in <<>>

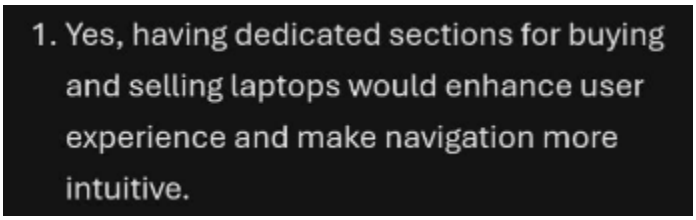
Requirements/Analysis

Week 2

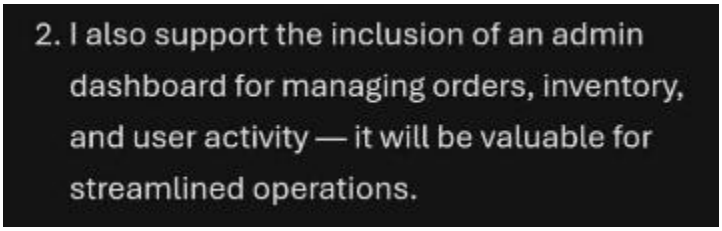
Journal

The following prompts are meant to aid your thought process as you complete the requirements/analysis portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

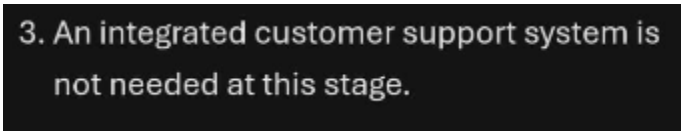
- After reading the client's brief (possibly incomplete description), write one sentence that describes the project (expected software) and list the already known requirements.
    - To develop a user friendly website for LapBoost to showcase their products and services.
        - The website must work in both mobile and laptop
        - Website must display all their products, services, etc.
        - Website should help in managing orders
- After reading the client's brief (possibly incomplete description), what questions do you have for the client? Are there any pieces that are unclear? After you have a list of questions, raise your hand and ask the client (your instructor) the questions; make sure to document his/her answers.
    - Should there be a separate section for buying and selling laptops?

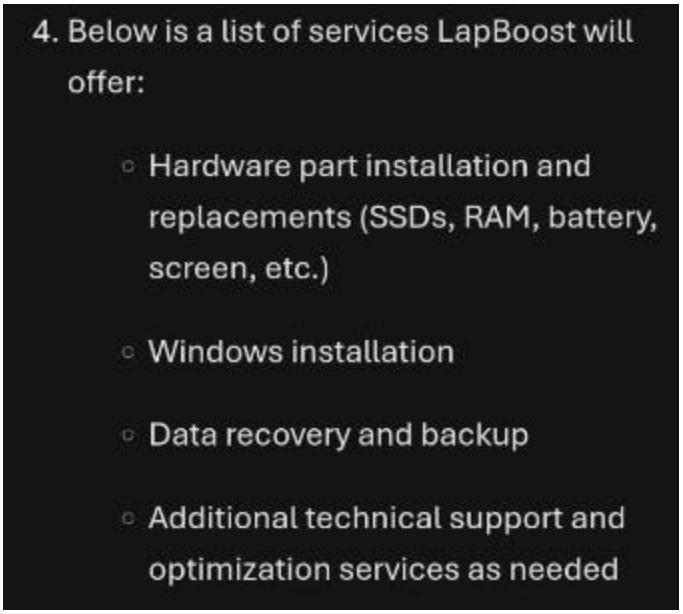
    - Does LapBoost need an admin dashboard to manage orders?

    - Should we include a customer support feature?

    - Tell about all the services and its description?



- Does the project cover topics you are unfamiliar with? If so, look up the topics and list your references.
    - To handle backend, Node.js is used which was a bit unfamiliar. Refer: https://www.w3schools.com/nodejs/
- Describe the users of this software (e.g., small child, high school teacher who is taking attendance).
    - All users who use laptops, who need services for laptops, who need buy parts of laptops.
- Describe how each user would interact with the software
    - Users will visit the website through lap or mobile browsers, and purchase their services.
- What features must the software have? What should the users be able to do?
    - View services offered
    - Submit service requests online
    - Contact LapBoost through a form

- Other notes:

Software Requirements

Software Description:
LapBoost is a startup that provides comprehensive laptop services, including repairs, upgrades, personalization, and buying/selling of laptops, with a focus on student-friendly, on-campus support. However, the company currently lacks an optimized online platform to efficiently showcase its services, streamline customer interactions, and manage service requests.
A well-designed website can enhance LapBoost's digital presence, improve customer engagement, and automate service booking, inquiries, and order management. The goal of this project is to develop a user-friendly, responsive, and feature-rich website that aligns with LapBoost's business model, ensuring easy access to their services for customers while improving operational efficiency.

Functional Requirement:

1. The system shall allow users to view available laptop services and products.
2. The system shall allow users to submit service requests through an online form.
3. The system shall allow users to contact LapBoost through a support/contact form.
4. The system shall allow administrators to access a dashboard to manage incoming service requests and products.
5. The system shall differentiate between regular users and administrators.
6. The system shall validate user input before processing forms.

Non Functional Requirement:

1. Usability:
   - The website should be easy to navigate, with a clear layout and understandable flow for both users and admin.
2. Performance:
   - The system shall respond to user actions within 2 seconds under normal conditions.
3. Scalability:
   - The design should support future expansion, such as adding features like user accounts, order tracking, or integrated payment processing.
4. Maintainability:
   - The system design shall ensure that features can be updated or modified with minimal effort.
5. Security:
   - User data and form submissions shall be protected against common threats such as injection attacks or unauthorized access.
6. Availability:
   - The website shall be designed to be available at all times during business hours and handle multiple users concurrently without crashing.

Black-Box Testing

Instructions: Week 4

Journal

Remember: Black box tests should only be based on your requirements and should work independent of design.

The following prompts are meant to aid your thought process as you complete the black box testing portion of this exercise. Please review your list of requirements and respond to each of the prompts below. Feel free to add additional notes.

- What does input for the software look like (e.g., what type of data, how many pieces of data)?
    - The inputs include text data such as user name and service description, and selected options like service type from a dropdown. These are typically 2–3 fields per form submission.
- What does output for the software look like (e.g., what type of data, how many pieces of data)?
    - The outputs are textual responses such as confirmation or error messages. In the admin panel, the output includes lists of submitted service requests and their statuses.
- What equivalence classes can the input be broken into?
    - Valid form submissions with all required fields filled
    - Invalid submissions with missing or empty fields
    - Submissions with special characters
    - Repeated submissions of the same data
- What boundary values exist for the input?
    - Name and service type must not be empty (minimum 1 character)
    - Description field has a practical limit around 255 characters
    - Whitespace-only input considered invalid
- Are there other cases that must be tested to test all requirements?
    - Accessing the admin page without proper credentials
    - Submitting the same request multiple times quickly
- Other notes:

Black-box Test Cases

Use your notes from above to complete the black-box test plan section of the formal documentation by writing black box test cases (other than actual results since no program currently exists). Remember to test each equivalence class, boundary value, and requirement.

Test ID	Description	Expected Results	Actual Results
1	Submit service request with valid name, service type, and description	Confirmation message is displayed, data is accepted	Works Properly
2	Submit service request form with all fields empty	Error message indicating required fields are missing	Works Properly
3	Attempt to access admin page without login	Access denied or redirected to login page	Cant be done, as the button is invisible
4	Register with valid username and password	Account created, redirected to login page	Works Properly
5	Login with valid credentials	Logged in, redirected to homepage	Works Properly
6	Login with incorrect password	Error message shown	Works Properly
7	Add product as admin	Product added successfully	Works Properly
8	Add product to cart	Item added to user’s cart	Works Properly
9	Checkout with cart items	Orders created, cart cleared	Works Properly
10	View current orders	List of active orders displayed	Works Properly
11	Admin accepts a pending service request	Status changes to "accepted"	Works Properly
12	Admin rejects a pending service request	Status changes to "rejected"	Works Properly
13	Buy another user’s listed item	Item removed from marketplace	Works Properly
14	Update username	Username changed, user logged out	Works Properly

Design

Instructions: Week 6

Journal

Remember: You still will not be writing code at this point in the process.

The following prompts are meant to aid your thought process as you complete the design portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

- List the nouns from your requirements/analysis documentation.
    - Users, services, products, orders, requests, forms, dashboard, laptop, admin, customer, message, device.
- Which nouns potentially may represent a class in your design?
    - User, Admin, Customer, ServiceRequest, Product, Order, Cart.
- Which nouns potentially may represent attributes/fields in your design? Also list the class each attribute/field would be a part of.
    - User: name, role, deviceType
    - ServiceRequest: serviceType, message, status
    - Product: productName, price, availability
    - Order: orderId, customerName, status
    - Cart: items, totalAmount
- Now that you have a list of possible classes, consider different design options (***lists of classes and attributes***) along with the pros and cons of each. We often do not come up with the best design on our first attempt. Also consider whether any needed classes are missing. These two design options should not be GUI vs. non-GUI; instead you need to include the classes and attributes for each design. Reminder: Each design must include at least two classes that define object types.
    - Option 1: Unified User Class with Role Tag
      Classes: User, ServiceRequest, Product, Order, Cart
      Pros:
        Simpler structure with fewer classes
        Easy to implement basic login and interaction
      Cons:
        Less flexibility for role-specific behaviors (admin vs. customer)
        Role logic handled through if/else conditions, increasing complexity later
    - Option 2: Separate Admin and Customer Classes inheriting from User (Chosen Design)
      Classes: User (abstract), Admin, Customer, ServiceRequest, Product, Order, Cart
      Pros:
        Clear separation of admin and customer responsibilities
        Easier to manage role-based access and behaviors
      Cons:
        Slightly more complex structure
        Requires proper inheritance handling
- Which design do you plan to use? Explain why you have chosen this design.
    We chose Option 2 because it provides better modularity and separation of concerns. It will make future extensions (like adding technician or delivery roles) easier to manage and helps avoid complex conditionals within a single user class.
- List the verbs from your requirements/analysis documentation.
    - View

    - Submit
    - Manage
    - Access
    - Contact
    - Fill
    - Browse
    - Interact
    - adjust.
- Which verbs potentially may represent a method in your design? Also list the class each method would be part of.
    - User: login(), logout()
    - Customer: submitRequest(), browseServices(), viewProducts()
    - Admin: manageRequests(), accessDashboard(), updateRequestStatus()
    - ServiceRequest: createRequest(), cancelRequest()
    - Product: checkAvailability(), updateStock()
    - Cart: addItem(), removeItem(), calculateTotal()
- Other notes:

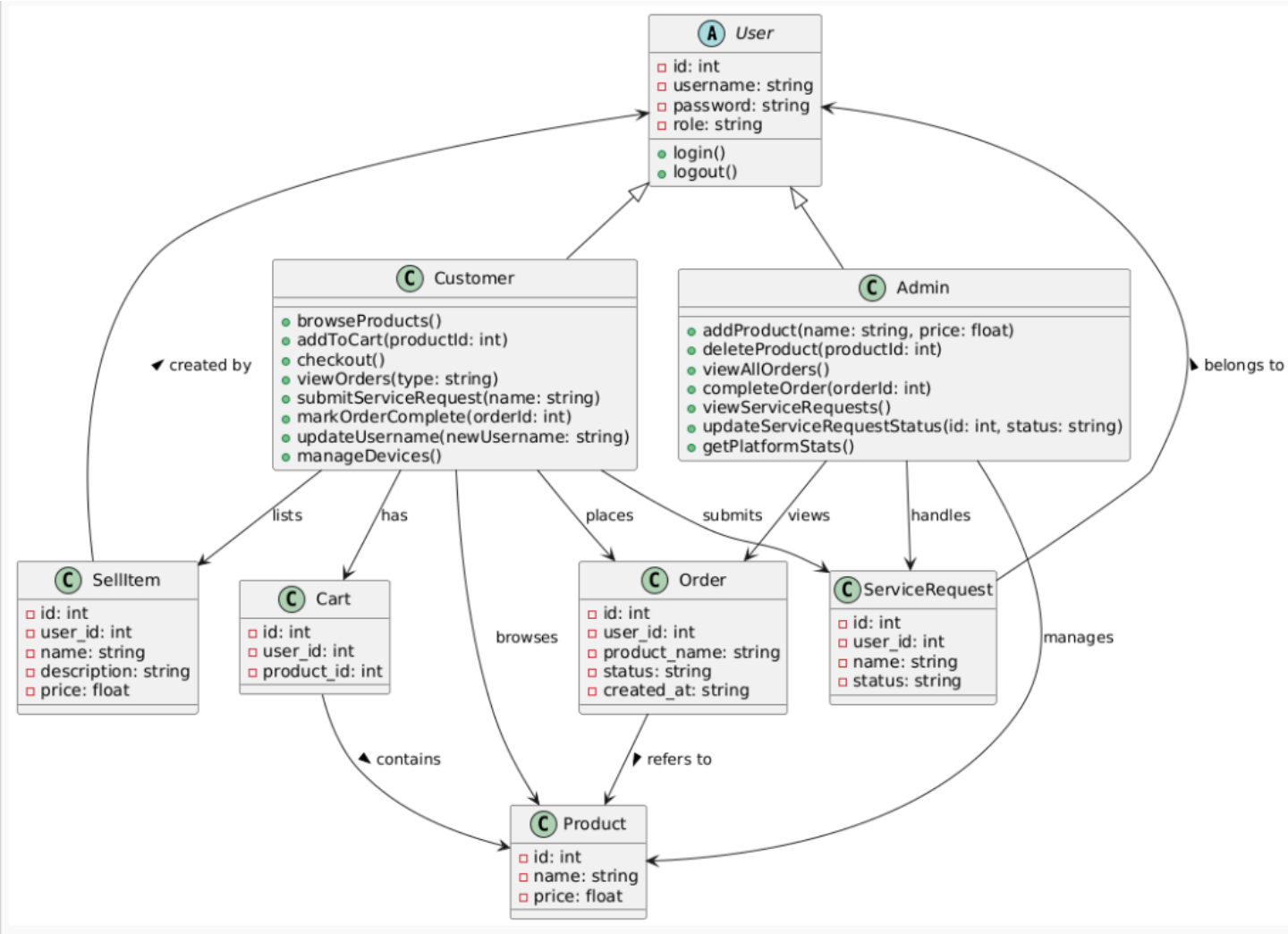
Software Design

Key Classes and Fields

1. User (Abstract)
    - Fields: id, username, password, role
    - Methods:
        - login(username, password)
        - logout()
2. Customer (extends User)
    - Methods:
        - browseProducts()
        - addToCart(product_id)
        - checkout()
        - viewOrders(type)
        - submitServiceRequest(service_name)
        - markOrderComplete(order_id)
        - updateUsername(newUsername)
        - manageDevices() (add/delete device)
3. Admin (extends User)
    - Methods:
        - addProduct(name, price)
        - deleteProduct(product_id)
        - viewAllOrders()
        - completeOrder(order_id)
        - viewServiceRequests()
        - updateServiceRequestStatus(id, status)
        - getPlatformStats()
4. Product
    - Fields: id, name, price
    - Methods:
        - getAllProducts()
        - createProduct(name, price)
        - deleteProduct(id)
5. Order
    - Fields: id, user_id, product_name, status, created_at
    - Methods:
        - createOrder(user_id, product_name)
        - getOrdersByStatus(user_id, status)
        - completeOrder(order_id)
6. Cart
    - Fields: id, user_id, product_id
    - Methods:
        - addItem(user_id, product_id)
        - getCart(user_id)
        - checkout(user_id)
        - clearCart(user_id)
7. ServiceRequest
    - Fields: id, user_id, name, status
    - Methods:

    - createRequest(user_id, name)
    - getAllRequests()
    - updateStatus(id, status)

8. SellItem
    - Fields: id, user_id, name, description, price
    - Methods:
        - listItem(user_id, name, desc, price)
        - getUserListings(user_id)
        - getAllListings()
        - buyItem(item_id, buyer_id)



For Customer
function checkout(userId) { ... }
function addToCart(userId, productId) { ... }

For Admin
function completeOrder(orderId) { ... }
function addProduct(name, price) { ... }

For General APIs
function login(username, password) { ... }

```
function logout(session) { ... }
```

Implementation

Instructions: Week 8

Journal

The following prompts are meant to aid your thought process as you complete the implementation portion of this exercise. Please respond to each of the prompt below and feel free to add additional notes.

- What programming concepts from the course will you need to implement your design? Briefly explain how each will be used during implementation.
    - **Object-Oriented Programming (OOP)**

      We designed the backend using OOP principles conceptually, separating responsibilities across user roles (Admin, Customer) and features (Order, Cart, Product, etc.).

      Although JavaScript and SQLite are not inherently object-oriented in structure, we followed the spirit of OOP in API design and data modeling.
    - **Client-Server Architecture**

      The project follows a clear client-server model:

      The **server (Node.js + Express)** handles routing, logic, and database operations.

      The **client (HTML + JavaScript)** sends requests and renders dynamic UI based on API responses.
    - **HTTP and REST APIs**

      We implemented RESTful API endpoints (/api/login, /api/products, /api/orders, etc.) to enable communication between frontend and backend.

      All major operations (login, order placing, service requests) are triggered via HTTP GET, POST, or DELETE.
    - **Session Management**

      Using express-session, we tracked logged-in users and managed authentication and role-based access control.

      Session data was used to validate actions (e.g., only admins can access certain routes).
    - **Database Management with SQL**

      We used SQLite as our persistent storage, creating normalized tables for users, products, orders, cart, devices, etc.

      We executed SQL commands to insert, update, delete, and query data.
    - **Form Handling and Input Validation**

      On the frontend, we collected user inputs using forms and validated them with both HTML constraints and JavaScript.

      On the backend, we validated request bodies and handled incorrect or missing data gracefully.
    - **JavaScript DOM Manipulation and Event Handling**

      Used JavaScript (main.js and inline scripts) to update the DOM dynamically based on API responses.

      Implemented event handlers for buttons like "Add to Cart", "Submit Service Request", and "Logout".
    - **Conditional Rendering and Role Checks**

      UI elements (e.g., Admin Panel link) and certain backend features were conditionally shown/accessible based on the role stored in the session.
- Other notes:

Implementation Details

Authentication Flow

- Users can register via register.html by entering a username and password.
- After registering, they can log in through auth.html.
- Session-based authentication ensures users remain logged in until logout.

User Functionalities

1. Homepage (home.html)

- Shows a welcome message and user's role.
- Displays a section for managing personal laptop devices.
- Includes a brief description of the company and a shortcut to services.

2. Marketplace (marketplace.html)

- Displays two sections:
  - Official products listed by LapBoost.
  - User-posted items for sale.
- Logged-in users can click "Buy" to purchase listed items or add official products to their cart.

3. Sell Devices (sell.html)

- Users can list devices for sale by filling out a form with:
  - Device name
  - Description
  - Price
- Their listings are displayed on the same page.

4. Cart (cart.html)

- Displays products the user has added to their cart.
- A "Checkout" button converts cart items into orders.

5. Orders (orders.html)

- Users can view their:
  - Current Orders (in progress)
  - Past Orders (completed)
- A "Mark as Completed" button is available for active orders.

6. Services (services.html)

- Lists predefined service types (e.g., RAM upgrade, Windows installation).
- Users can click "Request" to submit a service request.

7. Profile/Settings (settings.html)

- Displays:
  - Username
  - Role
  - Total, current, and completed orders
- Includes a form to update the username (which logs the user out after update).

Admin Functionalities (admin.html)

Admins have access to an additional dashboard that allows them to:

- View platform stats (total users, products, orders)
- Add/delete products for official sale
- View and manage all user orders
  - Mark orders as completed
- View and respond to service requests
  - Accept or reject service requests

Only users with the admin role can access this page. Others are redirected or denied.

Backend API (Powered by Node.js + Express)

The frontend communicates with the server via REST API routes, such as:

- POST /api/login – Login
- POST /api/register – Register new users
- POST /api/cart – Add to cart
- POST /api/checkout – Finalize cart into orders
- GET /api/orders/current – View current orders
- GET /api/products – Load all products
- POST /api/service/request – Submit a service request
- POST /api/profile/username – Update profile

All user state is stored via session. API calls are protected based on login status and user role (where applicable).

Database Initialization

The database is built using SQLite with the following tables:

- users, products, orders, cart
- devices, service_requests, sell_items

An init.js script is provided to clear and reseed initial data (e.g., test products).

Testing

Instructions: Week 10

Journal

The following prompts are meant to aid your thought process as you complete the testing portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

- Have you changed any requirements since you completed the black box test plan? If so, list changes below and update your black-box test plan appropriately.
    - User-to-user selling (SellItem): Users can now list devices for sale and others can buy them.
    - Admin order management: Admins can view and complete all user orders.
    - Username update: Users can change their username through the settings page.
    - Device registration: Users can register personal devices on the homepage.
- List the classes of your implementation. For each class, list equivalence classes, boundary values, and paths through code that you should test.
    - Class: User / Session
        - Equivalence classes:
          Valid login vs. invalid login
          Registered user vs. new registration
          User vs. admin role
        - Boundary values:
          Empty username or password
          Password with fewer than 6 characters
        - Tests:
          Login with correct credentials → success
          Login with wrong password → error
          Register with existing username → error
          Update username to a blank value → fail
    - Class: Product
        - Equivalence classes:
          Admin adding valid product vs. invalid/missing fields
          Users attempting restricted access
        - Boundary values:
          Empty product name or price = 0
        - Tests:
          Add product as admin → success
          Add product as user → access denied
          Delete non-existent product → error
          Load product list → correct data shown
    - Class: Cart
        - Equivalence classes:
          Add valid product vs. invalid product ID
          Checkout with items vs. empty cart
        - Boundary values:
          Repeated adds or large number of items
        - Tests:
          Add item to cart → visible in cart
          Checkout → items moved to orders

Cart clears after checkout → empty cart

- Class: Order
  - Equivalence classes:
    Current vs. completed orders
    Admin-completed vs. user-completed
  - Boundary values:
    Complete non-existent order → fail
  - Tests:
    User marks order as complete → success
    Admin completes any order → success
    View current/past orders → returns expected list
- Class: ServiceRequest
  - Equivalence classes:
    Valid request vs. empty service name
    Admin accepts vs. rejects
  - Boundary values:
    Long service names or duplicate submissions
  - Tests:
    Submit request → confirmation shown
    Admin accepts/rejects → status updates
    View requests by status → returns correct entries
- Class: SellItem
  - Equivalence classes:
    List own item vs. view all
    Buy others' listings vs. buy own
  - Boundary values:
    Empty name or negative price
  - Tests:
    Sell an item → appears in user listings
    Buy another user's item → purchase confirmed
    Attempt to buy own item → should be blocked
- Class: Device
  - Equivalence classes:
    Valid device name vs. blank name
    Add/delete devices as logged-in user
  - Boundary values:
    Duplicate device names or excessive device entries
  - Tests:
    Add device → added to list
    Delete device → removed from list
    View devices → lists only current user's devices

- Other notes:

Testing Details

Testing programs created to validate the system:

1. Register User Test – Checks whether a user can successfully register with valid credentials and handles duplicate username errors.
2. Login Test – Verifies successful login with correct credentials and appropriate error messages for incorrect passwords.
3. Add Product Test (Admin) – Confirms that an admin user can add new products and that the product appears in the official listings.
4. Access Control Test (Non-Admin) – Validates that a regular user cannot access admin-only features such as adding products.
5. Cart Management Test – Ensures that users can add products to their cart and view them before checkout.
6. Checkout Test – Checks that products in the cart are converted into orders and the cart is cleared afterward.
7. Order Management Test (User) – Verifies that users can view their current and past orders and mark their own orders as completed.
8. Order Management Test (Admin) – Ensures that the admin can view all user orders and mark any order as completed.
9. Service Request Submission Test – Tests the ability to submit a service request and handles invalid inputs like blank fields.
10. Service Request Management Test (Admin) – Checks whether an admin can accept or reject incoming service requests and updates the status accordingly.
11. Sell Item Test – Confirms that users can list items for sale with a name, description, and price, and that these listings appear correctly.
12. Buy Listing Test – Verifies that users can buy items posted by others, and that the listing is removed or updated after purchase.
13. Profile Update Test – Ensures that users can update their username and are logged out afterward for changes to take effect.
14. Device Management Test – Validates the ability to add and remove personal devices on the homepage.
15. Session and Role Test – Tests that correct session data (username, role, order stats) is returned and controls what content is shown on the frontend.

Presentation

Instructions:Week 12

Preparation

The following prompts are meant to aid your thought process as you complete the presentation portion of this exercise. It is recommended that you examine the previous sections of the journal and your reflections as you work on the presentation as it is likely that you have already answered some of the following prompts elsewhere. Please respond to each of the prompts below and feel free to add additional notes.

- Give a brief description of your final project
    - Our final project is LapBoost — a full-stack web platform that allows users to request laptop services, browse and purchase official products, sell their own devices, and manage orders online. The system supports both user and admin roles and includes full session management, order workflows, and role-based access control.
- Describe your requirement assumptions/additions.
    - Admins should manage orders and products through a secure dashboard.
    - Users should be able to sell and buy used devices in addition to browsing official products.
    - Each user may want to track their personal laptop devices.
    - Users may need to update their profile (e.g., username).
- Describe your design options and decision. How did you weigh the pros and cons of the different designs to make your decision?
    - Option 1: A single User class with role handling via conditional logic.
    - Option 2: A base User class with separate Admin and Customer roles extending it.

    We chose Option 2 for better separation of responsibilities and easier scalability. Although more complex, it allowed cleaner role-based API logic and made future expansions more manageable
- How did the extension affect your design?
    - The added features such as selling, admin order management, and service status tracking required new database tables, routes, and frontend elements. It made our backend logic more modular, and our frontend needed dynamic role-based controls to manage content visibility.
- Describe your tests (e.g., what you tested, equivalence classes).
    - Valid and invalid inputs (e.g., login credentials, empty forms).
    - Edge cases like repeated adds, empty carts, and invalid IDs.
    - Role-based access to protected routes.
    - State transitions such as marking orders completed or changing usernames.
- What lessons did you learn from the comprehensive exercise (i.e., programming concepts, software process)?
    - Build and connect a full-stack system using Node.js, Express, SQLite, and HTML/CSS/JS.
    - Design and test RESTful APIs.
    - Handle authentication, role management, and session storage.
    - Use OOP concepts in the design and structure of web applications.
    - Coordinate team development across frontend, backend, and database logic.
- What functionalities are you going to demo?
    - User login
    - Browsing and adding products to cart
    - Checking out and viewing orders
    - Selling and buying devices
    - Admin features: add/delete products, manage orders, respond to service requests

- Who is going to speak about each portion of your presentation? (Recall: Each group will have ten minutes to present their work; minimum length of group presentation is seven minutes. Each student must present for at least two minutes of the presentation.)
    - All demonstrations are done by Harikumaran Anandhan
    - Deployment process explained by Danush Vithiyarth jaiganesh
- Other notes: